dplyr package

- The dplyr package does not provide any new functionality, but it greatly simplifies existing functionality in R.
  - One important contribution of the dplyr package is that it provides a “grammar” (in particular, verbs) for data manipulation and for operating on data frames.
  - Another useful contribution is that the dplyr functions are very fast, as many key operations are coded in C++.

dplyr Grammer

- Some of the key “verbs” provided by dplyr packages are:
  - `select`: return a subset of the columns of a data frame, using a flexible notation.
  - `filter`: extract a subset of rows from a data frame based on logical conditions.
  - `arrange`: reorder rows of a data frame.
  - `rename`: rename variables in a data frame.
  - `mutate`: add new variables/columns or transform existing variables.
  - `summarise/summarize`: generate summary statistics of different variables in the data frame, possibly within strata.
  - `join`: Join two tables.
  - `%>%`: the “pipe” operator is used to connect multiple verb actions together into a pipeline.

Common dplyr function properties

- All of the functions will have a few common characteristics.
  - The first argument is a data frame.
  - The subsequent arguments describe what to do with the data frame specified in the first argument, and you can refer to columns in the data frame directly without using the $ operator (just use the column names).
  - The return result of a function is a new data frame.
  - Data frames must be properly formatted, that is, there should be one observation per row, and each column should represent a feature or characteristic of that observation.

```
filter(chicago, pm25tmean2 > 30)
```

dplyr package

- Install `dplyr` package.

  ```
  > install.packages("dplyr")
  ```

- For the example of dplyr, we will use a dataset contaning air pollution and temperature data for the city of Chicago.
- You can load the data into R using

  ```
  > chicago <- readRDS("chicago.rds")
  ```

- You can see some basic characteristics of the dataset with the `dim()` and `str()` functions.

```
> str(chicago)
'data.frame':   6940 obs. of  8 variables:
 $ city      : chr  "chic" "chic" "chic" "chic" ...
 $ tmpd      : num  31.5 33 33 29 32 40 34.5 29 ...
 $ dptp      : num  31.5 29.9 27.4 28.6 28.9 ...
 $ date      : Date, format: "1987-01-01" "1987-01-02" ...
 $ pm25tmean2: num  NA NA NA NA NA NA NA NA NA NA ...
 $ pm10tmean2: num  34 NA 34.2 47 NA ...
 $ o3tmean2  : num  4.25 3.3 3.33 4.38 4.75 ...
 $ no2tmean2 : num  20 23.2 23.8 30.4 30.3 ...
```

select()

- The `select()` function can be used to select columns(variables) of a data frame that you want to focus on.
- Suppose we wanted to take the first 3 columns only.

```
subset <- select(chicago, names(chicago)[1:3])
head(subset)
  city tmpd   dptp
1 chic 31.5 31.500
2 chic 33.0 29.875
3 chic 33.0 27.375
4 chic 29.0 28.625
5 chic 32.0 28.875
6 chic 40.0 35.125
```

- You can also do this by "`subset <- chicago[1:3,]`" but the dplyr function is much faster for a huge dataframe.
- The statement is equivalent to

```
subset <- select(chicago, city:dptp)
```

- You can also omit variables using the select() function by using the negative sign,

```
subset <- select(chicago, -(city:dptp))
```

- The `select()` function also allows a special syntax that allows you to specify variable names based on patterns.
- For example, if you wanted to keep every variable that ends with a “2”, we could do.

```
subset <- select(chicago, ends_with("2"))
str(subset)
> str(subset)
'data.frame':   6940 obs. of  4 variables:
 $ pm25tmean2: num  NA NA NA NA NA NA NA NA NA NA ...
 $ pm10tmean2: num  34 NA 34.2 47 NA ...
 $ o3tmean2  : num  4.25 3.3 3.33 4.38 4.75 ...
 $ no2tmean2 : num  20 23.2 23.8 30.4 30.3 ...
```

- Or if we wanted to keep every variable that starts with a “d”, we could do

```
subset <- select(chicago, starts_with(”d”))
str(subset)
’data.frame’:   6940 obs. of  2 variables:
 $ dptp: num  31.5 29.9 27.4 28.6 28.9 ...
 $ date: Date, format: ”1987-01-01” ”1987-01-02” ...
```

filter()

- The `filter()` function is used to extract subsets of rows from a data frame.
- This function is similar to the existing `subset()` function in R but is quite a bit faster.
- Suppose we wanted to extract the rows of the chicago data frame where the levels of PM2.5 are greater than 30.

```
chic.f <- filter(chicago, pm25tmean2 > 30)
str(chic.f)
'data.frame':   194 obs. of  8 variables:
 $ city      : chr  "chic" "chic" "chic" "chic" ...
 $ tmpd      : num  23 28 55 59 57 57 75 61 73 78 ...
 $ dptp      : num  21.9 25.8 51.3 53.7 52 56 65.8 ...
 $ date      : Date, format: "1998-01-17" "1998-01-23" ...
 $ pm25tmean2: num  38.1 34 39.4 35.4 33.3 ...
 $ pm10tmean2: num  32.5 38.7 34 28.5 35 ...
 $ o3tmean2  : num  3.18 1.75 10.79 14.3 20.66 ...
 $ no2tmean2 : num  25.3 29.4 25.3 31.4 26.8 ...
```

- You can place an arbitrarily complex logical sequence inside of `filter()`.
- For example, you can extract the rows where PM2.5 is greater than 30 and temperature is greater than 80 degrees Fahrenheit.

```
chic.f <- filter(chicago, pm25tmean2 > 30 & tmpd > 80)
select(chic.f, date, tmpd, pm25tmean2)
         date tmpd pm25tmean2
1  1998-08-23   81   39.60000
2  1998-09-06   81   31.50000
3  2001-07-20   82   32.30000
4  2001-08-01   84   43.70000
....
```

arrange()

- The `arrange()` function is used to reorder rows of a data frame according to one of the variables/columns.
- For example, you can order the rows of the data frame by date, so that the first row is the earliest (oldest) observation and the last row is the latest (most recent) observation.

```
chicago <- arrange(chicago, date)
head(select(chicago, date, pm25tmean2), 3)
        date pm25tmean2
1 1987-01-01         NA
2 1987-01-02         NA
3 1987-01-03         NA
```

- You can also do this using base R function `sort()`, but `arrange()` is easier and faster.
- Columns can be arranged in descending order too by using the special `desc()` operator.

```
chicago <- arrange(chicago, desc(date))
```

rename()

- Renaming a variable in a data frame is surprisingly hard to do!
- The `rename()` function is designed to make this process easier.
- The dptp column is supposed to represent the dew point temperature and the pm25tmean2 column provides the PM2.5 data. However, these names are pretty obscure or awkward and probably be renamed to something more sensible.

```
chicago <- rename(chicago, dewpoint = dptp, pm25 = pm25tmean2)
head(chicago[, 1:5], 3)
  city tmpd dewpoint       date     pm25
1 chic   35     30.1 2005-12-31 15.00000
2 chic   36     31.0 2005-12-30 15.05714
3 chic   35     29.4 2005-12-29  7.45000
```

- The syntax inside the rename() function is to have the new name on the left-hand side of the = sign and the old name on the right-hand side.
- Exercise: figure out how to do this in base R without dplyr.

mutate()

- The `mutate()` function exists to compute transformations of variables in a data frame.
- For example, with air pollution data, we often want to detrend the data by subtracting the mean from the data.
- That way we can look at whether a given day's air pollution level is higher than or less than average

```
chicago <- mutate(chicago, pm25detrend =
                      pm25 - mean(pm25, na.rm = T))
```

- There is also the related `transmute()` function, which does the same thing as `mutate()` but then drops all non-transformed variables.

```
chicago2 <- transmute(chicago,
 pm10detrend = pm10tmean2 - mean(pm10tmean2, na.rm = T),
 o3detrend = o3tmean2 - mean(o3tmean2, na.rm = T))
head(chicago2)
  pm10detrend  o3detrend
1  -10.395206 -16.904263
2  -14.695206 -16.401093
3  -10.395206 -12.640676
```

group_by()

- The `group_by()` function is used to generate summary statistics from the data frame within strata defined by a variable.
- For example, in this air pollution dataset, you might want to know what the average annual level of PM2.5 is.
    - So the stratum is the year, and that is something we can derive from the date variable

- To do this, create a year variable using `as.POSIXlt()`, a date-time Conversion Functions.
  - Create a time variable.

```
date=c(”2005-12-31”,”2003-12-30”)# character variable
date=as.POSIXlt(date)            # convert a time variable
date
date
[1] ”2005-12-31 KST” ”2003-12-30 KST” #
```

```
date$year  # extract year
[1] 105 103  # orgin 1900
```

```
date$year + 1900  # add origin
[1] 2005 2003
```

- Create a year variable

```
chicago <- mutate(chicago, year = as.POSIXlt(date)$year + 1900)
```

- Now create a separate data frame that splits the original data frame by year.

```
years <- group_by(chicago, year)
```

- In conjunction with the `group_by()` function we often use the `summarize()` function.

```
summarize(years, pm25 = mean(pm25, na.rm = TRUE),
                 o3 = max(o3tmean2, na.rm = TRUE),
                 no2 = median(no2tmean2, na.rm = TRUE))
    year     pm25       o3      no2
   <dbl>    <dbl>    <dbl>    <dbl>
 1  1987      NaN 62.96966 23.49369
 2  1988      NaN 61.67708 24.52296
 3  1989      NaN 59.72727 26.14062
 4  1990      NaN 52.22917 22.59583
```

- In a slightly more complicated example, we might want to know what are the average levels of ozone (o3) and nitrogen dioxide (no2) within quintiles of pm25.
  - First, create a categorical variable of pm25 divided into quantiles.

```
qq <- quantile(chicago$pm25, seq(0, 1, 0.2), na.rm = TRUE)
chicago <- mutate(chicago, pm25.quint = cut(pm25, qq))
```

  - Now group the data frame by the pm25.quint variable.

```
quint <- group_by(chicago, pm25.quint)
```

  - Finally, compute the mean of o3 and no2 within quintiles of pm25.

```
summarize(quint, o3 = mean(o3tmean2, na.rm = TRUE),
                 no2 = mean(no2tmean2, na.rm = TRUE))
   pm25.quint       o3      no2
       <fctr>    <dbl>    <dbl>
1   (1.7,8.7] 21.66401 17.99129
2  (8.7,12.4] 20.38248 22.13004
3 (12.4,16.7] 20.66160 24.35708
4 (16.7,22.6] 19.88122 27.27132
5 (22.6,61.5] 20.31775 29.64427
6          NA 18.79044 25.77585
```

Pipeline Operator

- The pipeline operator %>% is very handy for stringing together multiple dplyr functions in a sequence of operations
- Take the example that we just did in the last section where we computed the mean of o3 and no2 within quintiles of pm25.
  - create a new variable pm25.quint
  - split the data frame by that new variable
  - compute the mean of o3 and no2 in the sub-groups defined by pm25.quint
- That can be done in a single R expression.

```
mutate(chicago, pm25.quint = cut(pm25, qq)) %>%
       group_by(pm25.quint) %>%
       summarize(o3 = mean(o3tmean2, na.rm = TRUE),
       no2 = mean(no2tmean2, na.rm = TRUE))
```

- This way we don't have to create a set of temporary variables or create a massive nested sequence of function calls.